Python Basics

Table of Contents

1. Count Tuples in a List of Tuples

Given a list of tuples, find the number of occurrences of each tuple in the list.

In [34]:
```
def count_tuples(tuples_list):
    res = {}
    for tuple in tuples_list:
        if tuple in res.keys():
            res[tuple] +=1
        else:
            res[tuple] = 1
    return res

tuples_list = [(1,2,3), (1,2), (6,7,8), (1,2,3), (3,4), (1,2,3), (6,7,8)]

print(count_tuples(tuples_list))
```

```
{(1, 2, 3): 3, (1, 2): 1, (6, 7, 8): 2, (3, 4): 1}
```

2. Binary Search Trees in Python

Create a list of 10 random numbers between 1-100. Create a balanced BST (Binary search tree) from this array

In [35]:
```
import random

rand_list = [random.randint(1, 100) for i in range(10)]
rand_list
```

Out[35]:
```
[81, 84, 24, 90, 94, 97, 42, 99, 26, 55]
```

In [41]:
```
rand_list.sort()
print(rand_list)
class bst_node():
    def __init__(self, data):
        self.data = data
        self.right = None
        self.left = None

def sorted_array_to_BST(rand_list):
    if not rand_list:
        return None
    mid = len(rand_list)//2
    root = bst_node(rand_list[mid])
    root.left = sorted_array_to_BST(rand_list[:mid])
    root.right = sorted_array_to_BST(rand_list[mid + 1:])
    return root

def preorder_traversal(root):
    if not root:
        return
    print(root.data)
    preorder_traversal(root.left)
    preorder_traversal(root.right)

def height(root, contri = 0):
    if root is None:
        return contri
    return max(height(root.left, contri + 1), height(root.right, contri + 1))

def is_balanced(root):
    return abs(height(root.left, 0) - height(root.right, 0)) <= 1

root = sorted_array_to_BST(rand_list)
print("Preorder Traversal of constructed BST")
preorder_traversal(root)
print("Is the BST balanced?")
print(is_balanced(root))
```

```
[24, 26, 42, 55, 81, 84, 90, 94, 97, 99]
Preorder Traversal of constructed BST
84
42
26
24
81
55
97
94
90
99
Is the BST balanced?
True
```

3. List of Dictionary Keys

Given the following dictionary:

{ 'black :'r',
'hero':'e',
'go':'g',
'clue':'i',
'mean':'q',
'groan':'o',
'sin':'p',
'pint':'u',
'tone':'n',
'graze':'s',
'sea':'t',
'plant':'a'}

Create a list of all the keys of the dictionary, which can be formed from the values present in the dictionary.`

In [48]:
```
input_dict = {'black':'r',
              'hero':'e',
              'go': 'g',
              'clue':'i',
              'mean':'q',
              'groan':'o',
              'sin':'p',
              'pint':'u',
              'tone':'n',
              'graze':'s',
              'sea':'t',
              'plant':'a'}

def get_key_list(input_dict):
    value_dict = [input_dict[key] for key in input_dict.keys()]
    res = []
    for key in input_dict.keys():
        if all([True if char in value_dict else False for char in key]):
            res.append(key)

    return res

print(get_key_list(input_dict))
```

```
['go', 'groan', 'sin', 'pint', 'tone', 'sea']
```

4. Sorting List of Tuples

Given the following list of tuples,
a. Sort the list by 1st item in the tuple in ascending order
b. Sort the list by 2nd item in the tuple in ascending order
c. Repeat a, b using descending order

listA = [(1,2), (4,3), (2,10), (12, 5), (6, 7), (9,11), (15, 4)]

In [53]:
```
listA = [(1,2), (4,3), (2,10), (12, 5), (6, 7), (9,11), (15, 4)]

# a. Sort the list by 1st item in the tuple in ascending order
listA.sort(key = lambda x: x[0])
print(listA)

# b. Sort the list by 2nd item in the tuple in ascending order
listA.sort(key = lambda x: x[1])
print(listA)

# c. Repeat a, b using descending order
listA.sort(key = lambda x: x[0], reverse = True)
print(listA)

listA.sort(key = lambda x: x[1], reverse = True)
print(listA)
```

```
[(1, 2), (2, 10), (4, 3), (6, 7), (9, 11), (12, 5), (15, 4)]
[(1, 2), (4, 3), (15, 4), (12, 5), (6, 7), (2, 10), (9, 11)]
[(15, 4), (12, 5), (9, 11), (6, 7), (4, 3), (2, 10), (1, 2)]
[(9, 11), (2, 10), (6, 7), (12, 5), (15, 4), (4, 3), (1, 2)]
```

5. Mutable and Immutable Objects in Python

What do you mean by a mutable and immutable object? Are string, list, tuple, dictionary mutable type objects or immutable objects?

Any object whose internal states cannot be modified after creation is said to be immutable. The opposite is true for mutable objects.
Mutable Objects: Lists, dictionary
Immutable Objects: Strings, Tuple

6. Pandas and Numpy

Learn The usage of Pandas and Numpy: 1) Write the command to find unique values in education. 2) Command to find the number of customers subscribed and not subscribed. 3) Find the mean values of all the independent variables for every 'y' (outcome variable) 4) What is the mean age for every marital status 5) Check for null values 6) Find the descriptive statistics for each column 7) Use pd.query(), iloc, loc and range subsetting to extract insights from the data. You will be evaluated on the quality of your queries and the rationale behind selecting them. You can extract upto five (5) separate insights from the data.

In [1]:
```
import numpy as np
import pandas as pd

df = pd.read_csv('./Data/banking.csv')
df.head()
```

Out[1]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	poutc
0	44	blue-collar	married	basic.4y	unknown	yes	no	cellular	aug	thu	...	1	999	0	nonexi
1	53	technician	married	unknown	no	no	no	cellular	nov	fri	...	1	999	0	nonexi
2	28	management	single	university.degree	no	yes	no	cellular	jun	thu	...	3	6	2	su
3	39	services	married	high.school	no	no	no	cellular	apr	fri	...	2	999	0	nonexi
4	55	retired	married	basic.4y	no	yes	no	cellular	aug	fri	...	1	3	1	su

5 rows × 21 columns

In [60]:
```
# 1)    Write the command to find unique values in education.
print("Unique Values in Education Column: ", df.education.unique())
```

```
Unique Values in Education Column:  ['basic.4y' 'unknown' 'university.degree' 'high.school' 'basic.9y'
 'professional.course' 'basic.6y' 'illiterate']
```

In [66]:
```
# 2)    Command to find the number of customers subscribed and not subscribed.
print("Number of customers subscribed: ", df[df.y == 1].shape[0])
print("Number of customers not subscribed: ", df[df.y == 0].shape[0])
```

```
Number of customers subscribed:  4640
Number of customers not subscribed:  36548
```

In [73]:
```
# 3)    Find the mean values of all the independent variables for every 'y' (outcome variable)
print("Mean values of all independent variables for every outcome")
df.groupby('y').mean()
```

```
Mean values of all independent variables for every outcome
```

Out[73]:

	age	duration	campaign	pdays	previous	emp_var_rate	cons_price_idx	cons_conf_idx	euribor3m	nr_employed
y										
0	39.911185	220.844807	2.633085	984.113878	0.132374	0.248875	93.603757	-40.593097	3.811491	5176.166600
1	40.913147	553.191164	2.051724	792.035560	0.492672	-1.233448	93.354386	-39.789784	2.123135	5095.115991

In [75]:
```
# 4)    What is the mean age for every marital status
print('Mean age by marital status')
df.groupby(['marital'])['age'].mean()
```

```
Mean age by marital status
```

Out[75]:
```
marital
divorced    44.899393
married     42.307165
single      33.158714
unknown     40.275000
Name: age, dtype: float64
```

In [85]:
```
# 5)    Check for null values
print('Checking for null values...')
print(df.info())
print('-'*100)
print('No missing or null values. To confirm:-')
print('-'*100)
print([df[column].unique() for column in df.columns])
print('-'*100)
print("Yuppp....No null values")
```

```
Checking for null values...
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             41188 non-null  int64
 1   job             41188 non-null  object
 2   marital         41188 non-null  object
 3   education       41188 non-null  object
 4   default         41188 non-null  object
 5   housing         41188 non-null  object
 6   loan            41188 non-null  object
 7   contact         41188 non-null  object
 8   month           41188 non-null  object
 9   day_of_week     41188 non-null  object
 10  duration        41188 non-null  int64
 11  campaign        41188 non-null  int64
 12  pdays           41188 non-null  int64
 13  previous        41188 non-null  int64
 14  poutcome        41188 non-null  object
 15  emp_var_rate    41188 non-null  float64
 16  cons_price_idx  41188 non-null  float64
 17  cons_conf_idx   41188 non-null  float64
 18  euribor3m       41188 non-null  float64
 19  nr_employed     41188 non-null  float64
 20  y               41188 non-null  int64
dtypes: float64(5), int64(6), object(10)
memory usage: 6.6+ MB
None
----------------------------------------------------------------------------------------------------
No missing or null values. To confirm:-
----------------------------------------------------------------------------------------------------
[array([44, 53, 28, 39, 55, 30, 37, 36, 27, 34, 41, 33, 26, 52, 35, 40, 32,
       49, 38, 47, 46, 29, 54, 42, 72, 48, 43, 56, 31, 24, 68, 59, 50, 45,
       25, 57, 63, 58, 60, 64, 51, 23, 20, 74, 80, 61, 62, 75, 21, 82, 77,
       70, 76, 73, 66, 22, 71, 19, 79, 88, 65, 67, 81, 18, 84, 69, 98, 85,
       83, 78, 92, 86, 94, 17, 91, 89, 87, 95], dtype=int64), array(['blue-collar', 'technician', 'management',
'services', 'retired',
       'admin.', 'housemaid', 'unemployed', 'entrepreneur',
       'self-employed', 'unknown', 'student'], dtype=object), array(['married', 'single', 'divorced', 'unknow
n'], dtype=object), array(['basic.4y', 'unknown', 'university.degree', 'high.school',
       'basic.9y', 'professional.course', 'basic.6y', 'illiterate'],
      dtype=object), array(['unknown', 'no', 'yes'], dtype=object), array(['yes', 'no', 'unknown'], dtype=objec
t), array(['no', 'yes', 'unknown'], dtype=object), array(['cellular', 'telephone'], dtype=object), array(['au
g', 'nov', 'jun', 'apr', 'jul', 'may', 'oct', 'mar', 'sep',
       'dec'], dtype=object), array(['thu', 'fri', 'tue', 'mon', 'wed'], dtype=object), array([ 210,  138,  33
9, ..., 2260, 1662, 1490], dtype=int64), array([ 1,  3,  2,  8,  5,  4, 25, 11, 12, 18,  6, 17,  7, 20, 16, 14,
10,
        9, 19, 29, 13, 40, 15, 34, 22, 24, 41, 21, 23, 39, 28, 27, 31, 35,
       26, 30, 32, 43, 33, 42, 56, 37], dtype=int64), array([999,   6,   3,   2,   4,  16,   0,   5,  11,  14,
13,   9,  15,
        17,  12,  26,   7,  19,  10,   1,  18,   8,  20,  21,  22,  25,
        27], dtype=int64), array([0, 2, 1, 3, 4, 5, 7, 6], dtype=int64), array(['nonexistent', 'success', 'fail
ure'], dtype=object), array([ 1.4, -0.1, -1.7, -1.8, -2.9,  1.1, -3.4, -1.1, -3. , -0.2]), array([93.444, 93.2
, 94.055, 93.075, 92.201, 93.918, 92.893, 92.963,
       93.994, 94.465, 93.798, 92.431, 92.649, 92.843, 92.469, 93.749,
       93.876, 94.027, 94.199, 94.601, 92.713, 94.767, 93.369, 94.215,
       92.379, 92.756]), array([-36.1, -42. , -39.8, -47.1, -31.4, -42.7, -46.2, -40.8, -36.4,
       -41.8, -40.4, -26.9, -30.1, -50. , -33.6, -34.6, -40. , -38.3,
       -37.5, -49.5, -33. , -50.8, -34.8, -40.3, -29.8, -45.9]), array([4.963, 4.021, 0.729, 1.405, 0.869, 4.96
1, 1.327, 1.313, 1.266,
       1.41 , 4.864, 4.964, 4.965, 1.291, 4.96 , 4.962, 1.365, 4.86 ,
       4.967, 4.968, 1.344, 0.754, 1.299, 1.268, 1.334, 4.857, 0.715,
       4.966, 4.076, 1.354, 4.959, 4.958, 4.859, 1.27 , 4.856, 1.811,
       1.029, 1.259, 4.866, 0.883, 4.858, 1.56 , 0.74 , 4.245, 4.12 ,
       0.659, 1.415, 0.73 , 1.072, 4.153, 0.716, 0.682, 0.905, 1.281,
       4.865, 4.957, 0.914, 0.849, 0.876, 0.644, 4.855, 1.392, 1.25 ,
       0.873, 0.881, 0.942, 0.9  , 0.692, 1.244, 1.264, 4.191, 0.882,
       1.035, 0.742, 0.879, 1.032, 0.719, 5.   , 0.724, 4.97 , 0.646,
       1.26 , 1.479, 0.87 , 1.423, 1.498, 0.803, 0.714, 1.406, 0.702,
       0.827, 0.71 , 4.955, 0.653, 4.947, 0.835, 1.531, 0.735, 0.743,
       1.262, 0.944, 1.028, 0.663, 0.731, 0.699, 1.435, 1.538, 0.846,
       0.884, 1.453, 1.445, 0.635, 0.885, 0.854, 0.748, 0.643, 0.728,
       0.893, 0.861, 0.706, 1.025, 4.912, 0.668, 0.899, 1.085, 0.654,
       0.88 , 0.781, 1.059, 0.636, 0.965, 0.72 , 0.896, 1.602, 0.741,
       1.614, 0.84 , 1.483, 0.773, 0.701, 0.721, 0.697, 0.985, 0.829,
       0.761, 0.903, 1.466, 0.809, 1.049, 0.959, 0.717, 1.703, 1.04 ,
       0.898, 1.05 , 0.739, 1.   , 0.645, 1.007, 0.655, 1.252, 0.683,
       1.044, 0.651, 0.767, 0.797, 0.822, 0.904, 0.642, 0.752, 0.722,
       1.224, 0.977, 0.707, 1.548, 3.563, 1.016, 1.556, 1.663, 0.744,
       1.031, 0.851, 0.64 , 0.708, 0.878, 1.043, 0.886, 0.639, 0.793,
       0.788, 0.652, 4.794, 0.819, 1.206, 0.859, 1.687, 0.737, 4.956,
       1.799, 1.51 , 0.77 , 0.753, 0.877, 0.908, 1.041, 0.677, 0.723,
       0.937, 4.918, 0.987, 1.046, 4.936, 0.704, 0.712, 1.384, 0.711,
       0.821, 1.757, 0.695, 0.843, 1.52 , 0.838, 0.825, 4.7  , 1.03 ,
       1.048, 0.733, 4.474, 0.768, 0.982, 0.65 , 0.684, 1.037, 0.634,
       1.235, 0.921, 0.718, 0.685, 1.65 , 1.64 , 1.286, 0.732, 5.045,
       0.649, 1.4  , 4.663, 0.778, 0.672, 1.629, 1.372, 1.726, 0.75 ,
       0.688, 0.709, 0.713, 1.215, 0.889, 4.827, 0.834, 0.933, 3.743,
       0.727, 0.755, 1.584, 4.406, 0.89 , 0.81 , 0.972, 0.782, 0.638,
       1.039, 0.79 , 0.802, 3.853, 1.099, 0.7  , 0.762, 4.76 , 0.953,
       0.766, 0.637, 1.778, 0.771, 3.053, 3.329, 0.969, 0.813, 0.894,
       4.733, 0.895, 3.879, 4.286, 4.592, 3.901, 0.979, 0.891, 1.047,
       0.927, 1.018, 1.008, 1.045, 0.749, 0.888, 4.921, 0.993, 0.956,
       4.223, 0.69 , 0.996, 3.669, 1.574, 3.488, 3.428, 4.343, 3.816,
       3.282]), array([5228.1, 5195.8, 4991.6, 5099.1, 5076.2, 5191. , 5017.5, 5008.7,
       4963.6, 5023.5, 5176.3]), array([0, 1], dtype=int64)]
----------------------------------------------------------------------------------------------------
Yuppp....No null values
```

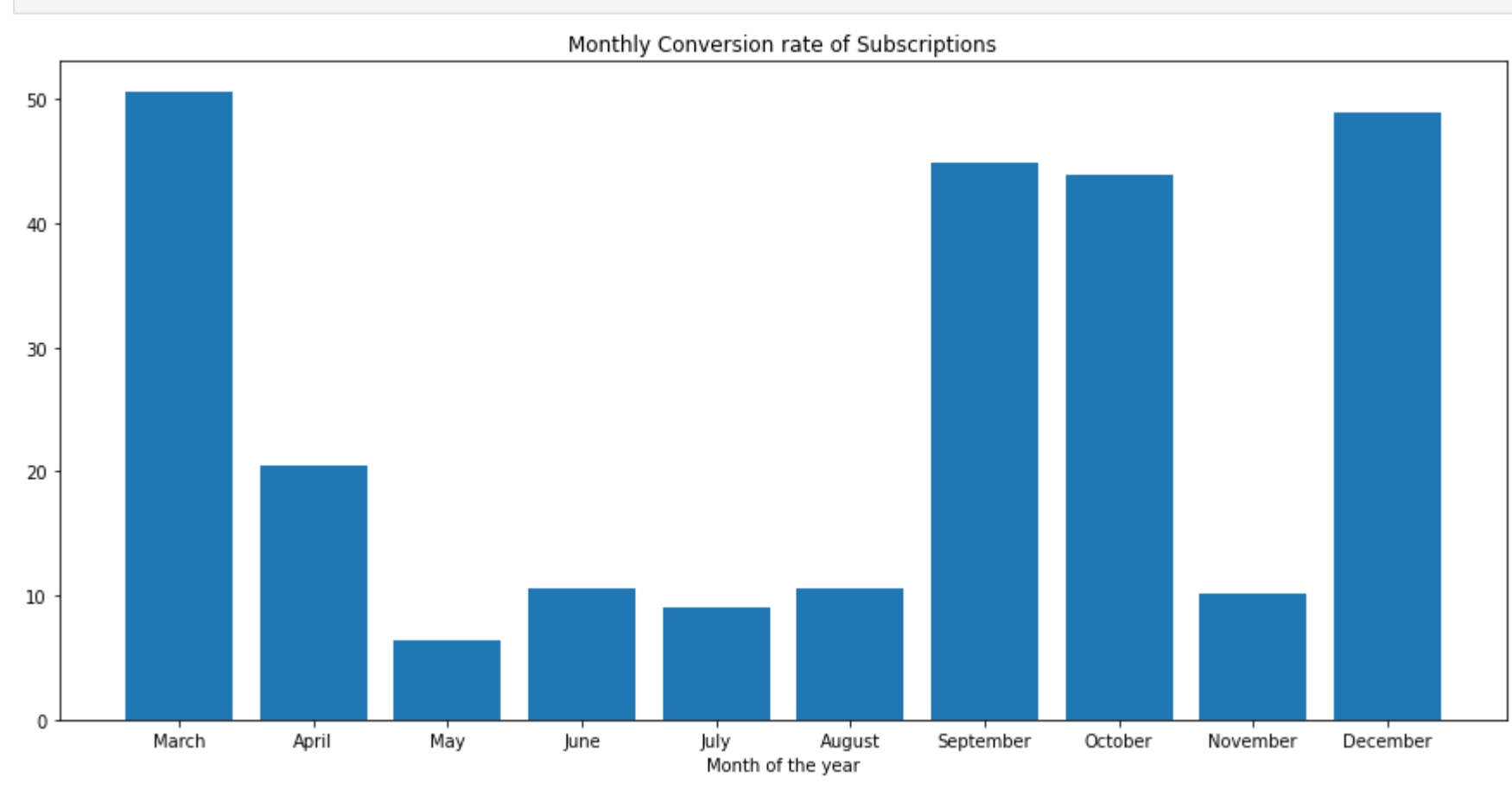
In [89]:
```
# 6)    Find the descriptive statistics for each column
print("Descriptive Statistics of columns")
df.describe()
```

```
Descriptive Statistics of columns
```

Out[89]:

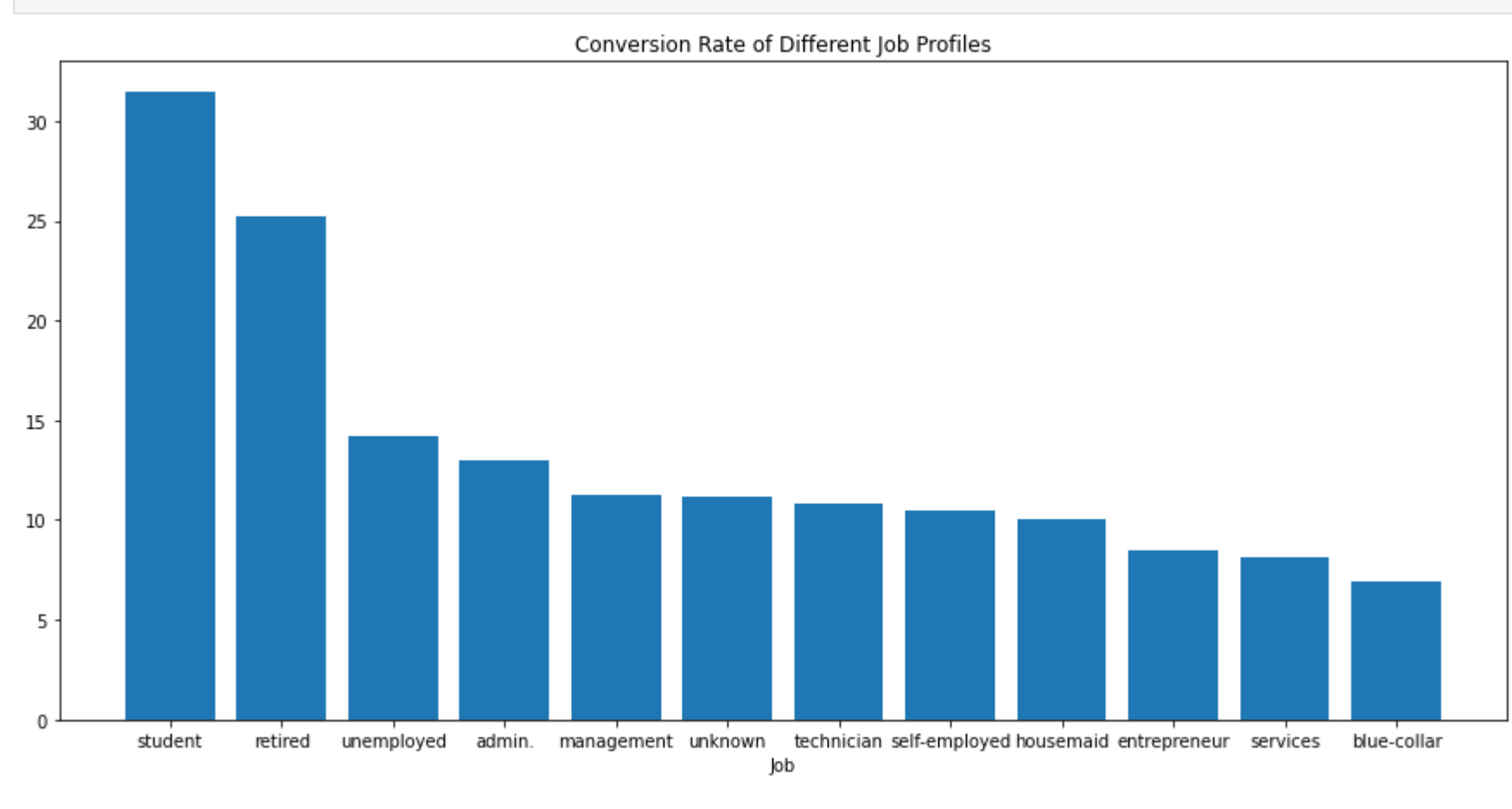
	age	duration	campaign	pdays	previous	emp_var_rate	cons_price_idx	cons_conf_idx	euribor3m	nr_emp
count	41188.00000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.0
mean	40.02406	258.285010	2.567593	962.475454	0.172963	0.081886	93.575664	-40.502600	3.621291	5167.0
std	10.42125	259.279249	2.770014	186.910907	0.494901	1.570960	0.578840	4.628198	1.734447	72.2
min	17.00000	0.000000	1.000000	0.000000	0.000000	-3.400000	92.201000	-50.800000	0.634000	4963.6
25%	32.00000	102.000000	1.000000	999.000000	0.000000	-1.800000	93.075000	-42.700000	1.344000	5099.1
50%	38.00000	180.000000	2.000000	999.000000	0.000000	1.100000	93.749000	-41.800000	4.857000	5191.0
75%	47.00000	319.000000	3.000000	999.000000	0.000000	1.400000	93.994000	-36.400000	4.961000	5228.1
max	98.00000	4918.000000	56.000000	999.000000	7.000000	1.400000	94.767000	-26.900000	5.045000	5228.1

In [59]:
```
# 7) Use pd.query(), iloc, loc and range subsetting to extract insights from the data.
#    You will be evaluated on the quality of your queries and the rationale behind selecting them.
#    You can extract upto five (5) separate insights from the data.
import matplotlib.pyplot as plt
%matplotlib inline

# Feature number 1:-
freq = df[df['y'] == 1].groupby(['month'])['y'].count()
freq_general = df.groupby(['month'])['y'].count()
freq = (freq/freq_general) * 100
new_index = ['mar','apr', 'may','jun', 'jul', 'aug', 'sep','oct','nov','dec']
freq = freq.reindex(new_index)

plt.figure(figsize = [15,7])
plt.bar(x = freq.index, height = freq)
plt.xlabel("Month of the year");
plt.xticks(range(10), ['March', 'April', 'May', 'June', 'July', 'August', 'September','October', 'November', 'D
plt.title("Monthly Conversion rate of Subscriptions");
```



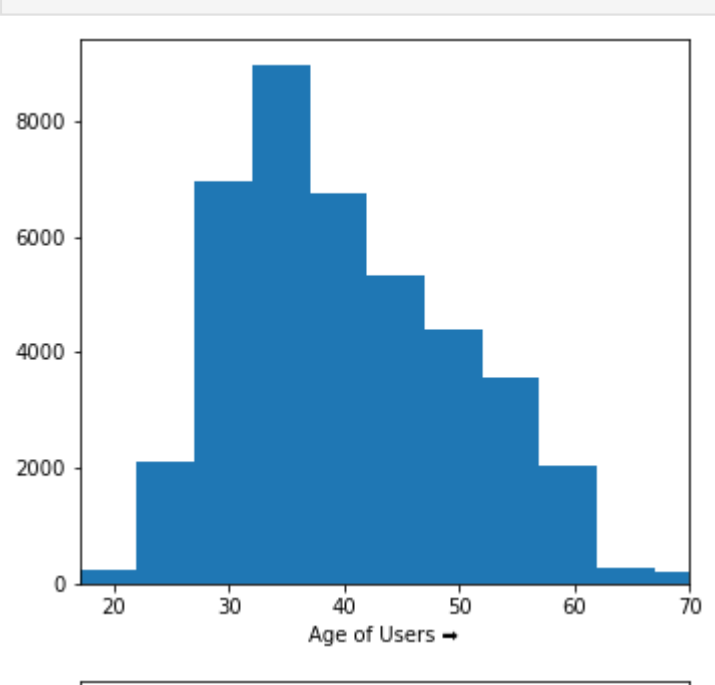
So as can be seen the subscriptions don't seem to be concentrated as per seasons and generally see high conversion values in March, September, October and December

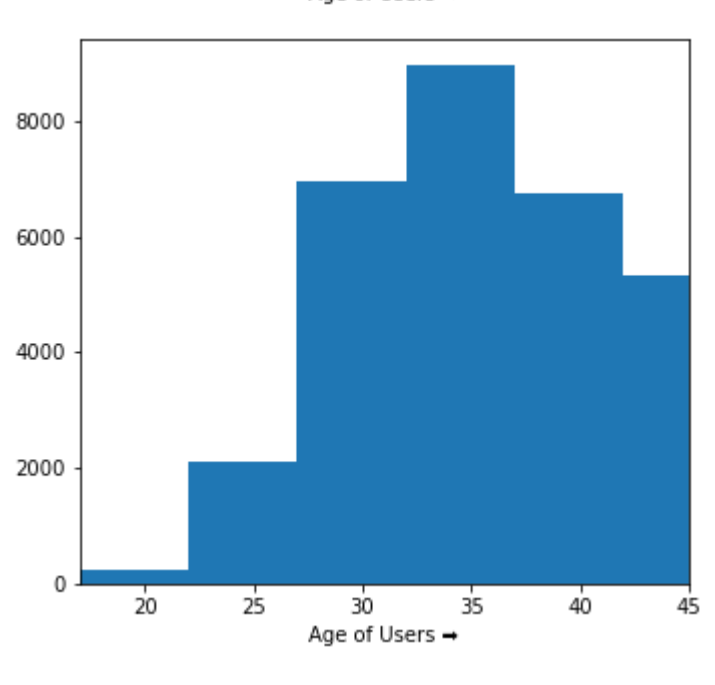
In [39]:
```
# Feature 2:- Distribution by job type
freq_by_job = df[df['y'] == 1].groupby('job')['y'].count()
freq_general = df.groupby(['job'])['y'].count()
freq_dist = (freq_by_job/freq_general)*100
freq_dist = freq_dist.sort_values(ascending = False)

plt.figure(figsize = [15,7])
plt.bar(x = freq_dist.index, height = freq_dist)
plt.xlabel("Job");
plt.xticks()
plt.title("Conversion Rate of Different Job Profiles");
```



We can see that the highest subscriptions are from classes that don't make money. So perhaps the subscriptions pertain to a product that benefits these groups

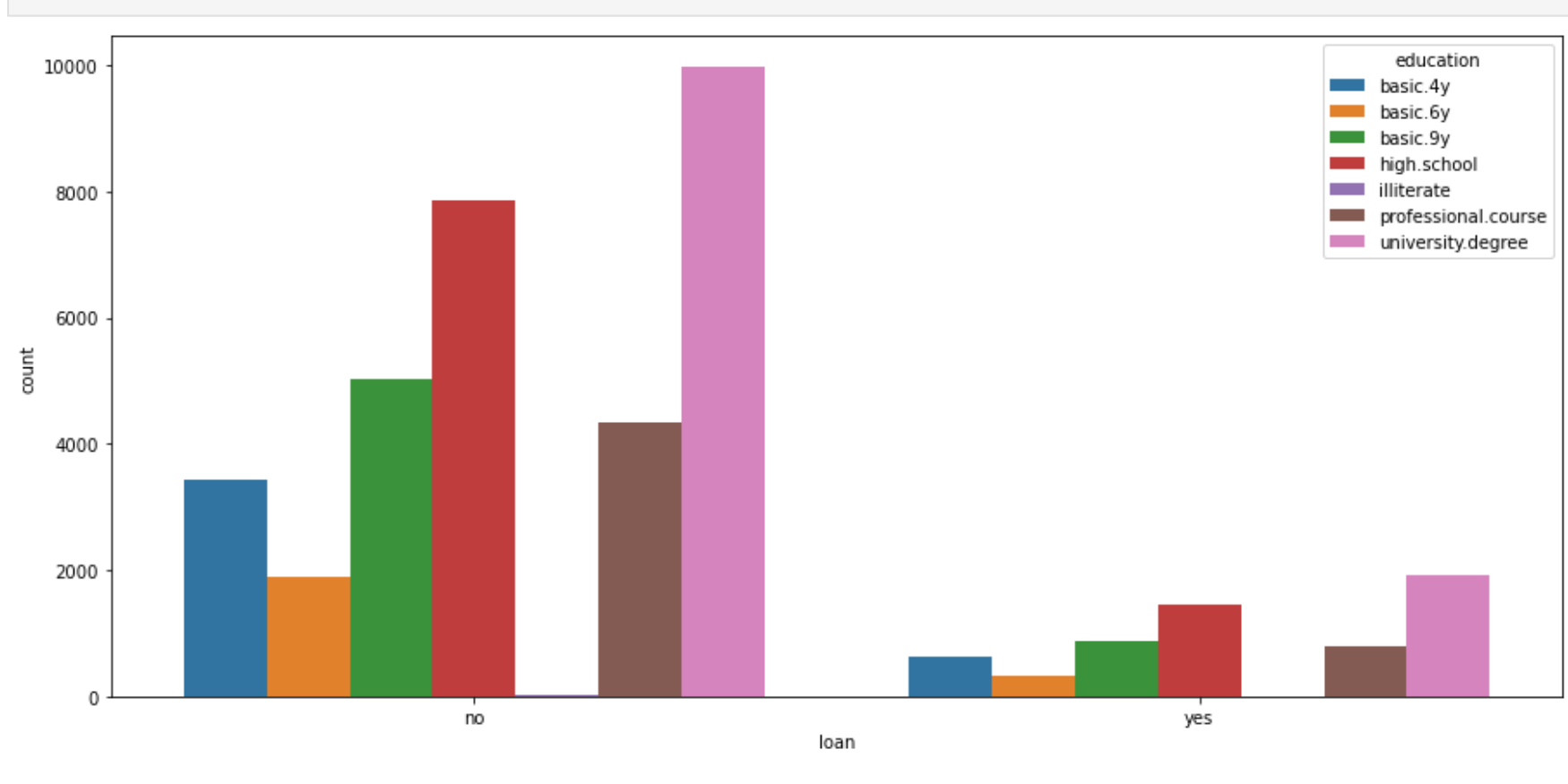
In [56]:
```
# Feature 3: Age group of the subscribers
bins = np.arange(df.age.min(), df.age.max()+5, 5)
df['age_bin'] = pd.cut(df.age, bins)

def histogram(df, x, bins, xlim_upper, xlabel):
    """Function for plotting histogram with parameters and ticks defined"""
    plt.hist(data=df, x = x, bins = bins);
    plt.xlim(df.age.min(), xlim_upper);
    plt.xlabel(xlabel);

plt.figure(figsize = [12, 5])
plt.subplot(1,2,1)
histogram(df, 'age', bins, 70, 'Age of Users ➡')

plt.figure(figsize = [12, 5])
plt.subplot(1,2,2)
histogram(df, 'age', bins, 45, 'Age of Users ➡')
```





Turns out, a lot of the consumers are between 32 to 37 years of age

In [93]:
```
# Feature 4: Correlation between housing and loan

def clustered_bar_plot(size, freq_by_loan, x, y, hue):
    import seaborn as sb
    plt.figure(figsize = size)
    sb.barplot(data=freq_by_loan, x = x, y = y, hue = hue)

freq_by_loan = df[df['loan']!='unknown'].groupby(['loan', 'education'])
freq_by_loan = freq_by_loan.size().reset_index()
freq_by_loan.columns = ['loan','education','count']
freq_by_loan = freq_by_loan[freq_by_loan['education']!='unknown']
freq_by_loan.reset_index()
clustered_bar_plot([15,7], freq_by_loan, 'loan', 'count', 'education')
```



Seems like the educated guys aren't taking enough loans (in absolute numbers atleast)!!